Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 85 (2016) 929 – 939

Procedia Computer Science

Natural Computing for Automatic Test Data Generation Approach Using Spanning Tree Concepts

Sanjay Singla[a*], Dr. Raj Kumar[b], Dr. Dharminder Kumar[c]

[a] Research Scholar, UIET, MDU Rohtak
[b]Assistant Professor, MDU, Rohtak
[c] Professor, GJUS&T, Hisar

Abstract

The weight is assigned to all edges of graph according to a uniform policy and focus is on the leaves nodes of the Spanning Tree of Control Flow Graph of Program. If all leaves nodes of Spanning tree are covered by the test cases, then number of test cases required or testing the program can be reduced to great extent. Further the spanning tree is based on the concept of maximum spanning tree of CFG which helps in finding the critical paths of CFG. The chances of finding errors in critical are more as compare to the normal path. Covering of all the leaves nodes of Spanning Tree give guarantee for the coverage of Critical path also. Further the paper also shows that number of test cases required in case of Spanning tree is less as compare to the Dominance tree concept. For the generation of fast, unique and reliable test cases a optimization technique which is based on natural computing concept is used which is called as on Swine Influenza Models Based Optimization (SIMBO). The relation between nodes has been used as fitness function in this paper. Finally, the results in the paper show the effectiveness of SIMBO techniques as compare to Particle Swarm Optimization (PSO) and Genetic Algorithm (GA).

Keywords: Data flow testing, Particle Swarm Optimization (PSO), Genetic algorithm (GA), Swine Influenza Models Based Optimization (SIMBO)

1. Introduction

The most difficult and time consuming process in software life cycle which consumes 50% cost of software development process is software testing. The most difficult problem in testing is to find a valid and reliable test data strategy. To satisfy a given testing criteria, numerous test data has to be generated which required lots of time and efforts. The most critical task in software testing process is the designing of test cases for the satisfaction of given testing criteria [26]. They are different type of techniques or methods purposed by different researches/others from time to time to generate test data for software testing [5]-[8], [10]-[13], [30].

The cost, time and efforts for designing different test casing for testing the software can be reduced to large extend if the process of testing is done automatically without involvement human beings. There are various Software Computing Technique like Swarm Intelligence, Genetic algorithm, Evolutionary Programming etc which has shown their impact and influence in the field of software testing.[25]-[26]. The performance of all these Software Computing Algorithm is significant to large extent but time taken by these algorithms is relatively long for exploring and exploiting the promising reasons in the search space [31]. Genetic Algorithm (GA) is based on Darwin's theory of Natural Evolution specified in the origin of spices. It was purposed by Holland in 960. It is based on the concept of population of chromosome where is chromosome represents a corresponding solution. These solutions / chromosome undergo various operations to produce next generation. The operations used in Genetic Algorithm are crossover and mutation operator which helps the GA to evolve from generation to generation. Genetic algorithm is widely used in many Engineering and Optimization problem and it has proved its existence and influence but it is trapped in local optimization and it is time consuming.

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: dr.ssinglacs@gmail.com

Peer-review under responsibility of the Organizing Committee of CMS 2016
doi:10.1016/j.procs.2016.05.284

Kennedy and Eberhart have purposed a new population based optimization technique called Partial Swarm Optimization (PSO) in 1995. PSO is encouraged by social behavior of birds flocking and fish Schooling [17]. This new algorithm has some unique feature which makes this algorithm more better technique as compare to the Genetic algorithm. First, PSO used concept of memory which is not available in Genetic Algorithm. It keeps the information of each particle best position, that they achieved from the starting point of their journey and the global best position among all the particles. Due to this memory component, PSO moves faster towards the goal. This type of information is not stored anywhere in Genetic algorithm. Second, it is very easy to understand and implementation PSO as compare to Genetic algorithm. Third, there is information sharing among the particles in PSO through constructive cooperation which is lacking in Genetic algorithm [33]. Hence PSO performance betters as compare to the Genetic algorithm [22]. There are different number of research papers which prove that performance of PSO for solving a number of different type of software testing problems is better or is equally good than the performance of Genetic algorithm but there are two major issues associated with the PSO. First, It can be easily Trapped in local minima when some poor parties attract the other particle. Second, the performance of PSO is depend on the problem in hand [31]. To take over the issue of GA and PSO, a new optimization method called Swine Influenza Models Based Optimization (SIMBO) developed by S. S. Pattnaik is used. The optimization algorithm SIMBO uses a fitness function which uses relations concepts between nodes of spanning tree of program. Finally the comparison of performance of these algorithms on the basis of results is shown.

2. BACKGROUND

This unit describes the basic concept which has to be used throughout of the paper.

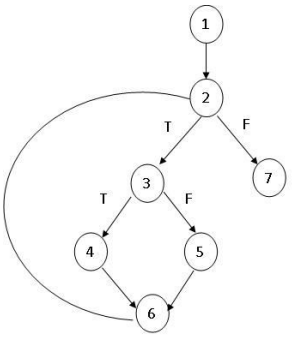
A. Control Flow Graph (CFG)

The program's CFG is a directed graph. In CFG, every node represents a group of consecutive statements, which together represent a basic block and every edge represents movement of control flow between the vertices. To build a CFG we first build basic blocks and then we add edges that represent control flow between these basic blocks. Figure1 shows the program whose CFG is shown in figure 2.

```
1   1   #include<stdio.h>
2   1   void main ()
3   1   {
4   1   int a, b , c;
5   1   a= b= 0;
6   1   scanf("%d", &c);
7   2   while (c <> 0)
8   2   {
9   3           if ( c %2 = = 0)
10  4           {
11  4           a = a + 1
12  4           }
13  5           else
14  5           {
15  5           b = b +1 ;
16  5           }
17  6   scanf("%d", &c);
18  6   }
19  7   printf ("%d   %d",  a , b);
20  7   }
```

Fig. 1. Program 1

A Directed graph or Digraph is represented as G (V, E) where V is the set of vertices given by V= {v_1, v_2 , …..}, and E is the set of edges given by E = {e_1, e_2 ,….}, and a mapping that maps every edge onto some ordered pair of vertices (v_i, v_j).



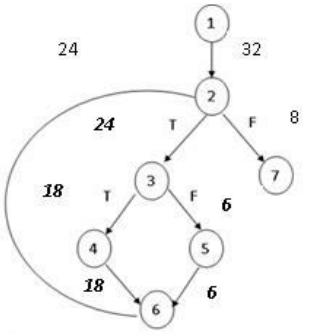


Fig. 2. Control Flow Graph

Fig. 3. Weighted Control Flow Graph

B. Weighted Graph

A graph G is said to be a weighted graph if its edges are assigned some weight. Weight is assigned to edges of graph. First, number of Level is counted and then weight is initiated as 2^n, where n is the number of Levels. In the figure 2, there are 5 levels so weight is initiated as $2^5 = 32$. This weight bifurcate on the decision node, 75% of the decision node weight moves towards the branch which supports the true decision of predicate node while 25% moves towards the opposite side. The weight for the CFG of Figure 2 is shown as in fig 3.

C. Adjacency Matrix of a Diagraph

Adjacency matrix is used to represent weighted graphs. An adjacency matrix (X) of a G with n vertices that are assumed to be ordered from v_1 to v_n is defined by

$$x_{ij} = 1, \text{ if there exist an edge between } v_i \text{ and } v_i$$
$$= 0, \text{ therewise.}$$

If $[x_{ij}]=w$, then there is an edge from vertex v_i to vertex v_j with weight w.
The adjacency matrix X of the figure 3 is shown as

X =	1	2	3	4	5	6	7
1	0	32	0	0	0	0	0
2	0	0	24	0	0	0	8
3	0	0	0	18	6	0	0
4	0	0	0	0	0	18	0
5	0	0	0	0	0	6	0
6	0	24	0	0	0	0	0
7	0	0	0	0	0	0	0

D. Spanning Tree

A tree T is said to be a spanning tree of a connected graph G if T is a subgraph of G and T contains all vertices of G. A spanning tree is a sort of skeleton of original Graph G. So Spanning tree is sometimes referred to as a skeleton or scaffolding of G. A spanning tree with the smallest weight in the weighted graph is called a shortest spanning tree or shortest distance spanning tree or minimal spanning tree. A minimal spanning tree in connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges. In this paper the modified Prim Algorithm is used to obtain minimal spanning tree, but instead of minimal spanning tree, maximum spanning tree (spanning tree having highest weight) is required. As Prim Algorithm has two limitations, *first it is not applicable if weights are negative and second it is used only for undirected graph but not for directed graph. For* maximum Spanning tree, first adjacency matrix is multiplied with -1.

Adjacency Matric X1 after Multiply X by -1 is as

X1 =	1	2	3	4	5	6	7
1	0	-32	0	0	0	0	0
2	0	0	-24	0	0	0	-8
3	0	0	0	-18	-6	0	0
4	0	0	0	0	0	-18	0
5	0	0	0	0	0	-6	0
6	0	-24	0	0	0	0	0
7	0	0	0	0	0	0	0

After that modified Prim algorithm is applied. As Prime algorithms starts from the First vertex v_1 and connect it to its nearest neighbor (i.e., to the vertex which has the smallest entry in row 1 of the table), say v_k Now consider v_1 and v_k as one subgraph and connect this subgraph to its closet neighbor (i.e., to the vertex other than v_1 and v_k that has the smallest enetry among all the entries among rows 1 and k). Repeat this process till all n nodes or vertices have been connected by n-1 edges.

The highlighted value is selected using Prim algorithm is shown as

			1	2	3	4	5	6	7
		1	0	**-32**	0	0	0	0	0
		2	0	0	**-24**	0	0	0	**-8**
		3	0	0	0	**-18**	**-6**	0	0
X1	=	4	0	0	0	0	0	**-18**	0
		5	0	0	0	0	0	-6	0
		6	0	-24	0	0	0	0	0
		7	0	0	0	0	0	0	0

All other value in the matrix except highlighted become zero which is shown as below

			1	2	3	4	5	6	7
		1	0	**-32**	0	0	0	0	0
		2	0	0	**-24**	0	0	0	**-8**
		3	0	0	0	**-18**	**-6**	0	0
X1	=	4	0	0	0	0	0	**-18**	0
		5	0	0	0	0	0	0	0
		6	0	0	0	0	0	0	0
		7	0	0	0	0	0	0	0

The rows which have all zero values are leave nodes, which is main concern to this paper. Figure 4 shows minimal spanning tree and fig 5 shows maximum spanning tree.

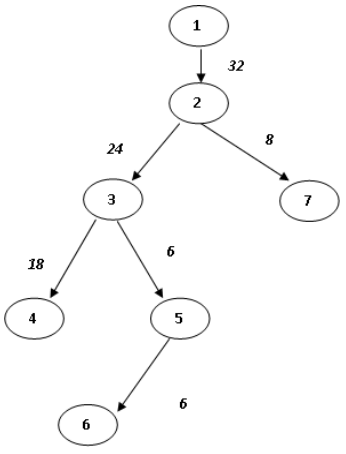


Fig. 4. Minimal Spanning Tree

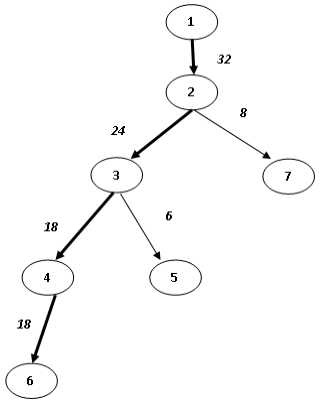


Fig. 5. Maximal Spanning Tree

E. Critical Path

Critical path has a major role in software testing because a lot of error falls in the critical path. If critical path is tested properly then the chances of occurring of errors in software can be reduced to large extent because the critical path has more chances to trap errors during software testing. In the maximum spanning tree, the path having highest weight is critical path. These paths must be analyzed during testing. The critical path is 1-2-3-4-6 which is shown by dark line.

F Reduce the Test Cases

The main aim of research is to cover all statement of program and find maximum errors with minimum number of test cases. This can be done by covering a subset of statements that guarantees the coverage of all statements of the tested program [29]. The main concern is the complete set of leaves nodes of maximum spanning tree as every set of path that covers it also covers all nodes in tree. The set of leave nodes for figure 5 is as X= [6, 5, 7]. The path of each and every element of set X is shown as.
path (6) = 1,2,3,4,6 - Critical path
path (5) = 1,2,3,5
path (7) = 1,2, 7

Further the covering of Path (6) covers automatically the critical path which is more prone to error. All nodes of the CFG (1, 2, 3, 4, 5, 6, 7 nodes of CFG shown in fig 2) can be covered by covering the path of each and every element from the set of leaf node of maximum spanning tree. Covering of each and every node of CFG means covering of all statements of program and this is main aim behind testing of software.
The dominance tree of CFG of fig 2 is shown in fig 6. The number of leaves nodes for Dominance Tree in figure 5 is 4 i.e. [4, 5, 6, 7] while for spanning tree, the number of leaves nodes is 3 [5, 6, 7]. Further as compare to the testing of leave nodes of dominance tree proposed by A S Ghiduk [29], the number of test cases requires is less in using maximum spanning tree, thus reduce the effort to a very large extent.

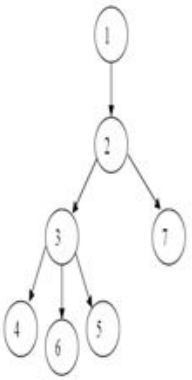


Fig. 6. Dominance Tree

III. SWINE INFLUENZA MODELS BASED OPTIMIZATION (SIMBO)

Swine Influenza Model Based Optimization (SIMBO) developed by Pattnaik et al. [1] is optimization method based on swine influenza, a diseaserelated with the respiratory tract.

A. Swine Influenza Model Based Optimization with Treatment (SIMBO-T)

Key Terms and Definitions
Day (D): current generation or iteration.
TD: total number of days or generations.
TI: total number of individuals in Population.
State (S): individual position.
Health (H): fitness.
Pandemic state (PS):
Pandemic (global) best state in all the individuals.
Pandemic health (PH): fitness value corresponding to pandemic state.
Momentum factor of dose (Md): it is for controlling the individual dose.
Momentum factor of state (MS): it is for controlling the individual state

Primary Symptom Per Day (Primary (Day)) the primary symptoms of swine flue caused due to fever, cough, fatigue and headache, nausea and vomiting and diarrhea during each day.
A mathematical expression is developed to relate these parameters. It is expressed by Eq. (1)

$$\text{Primary (Day)} = (\text{Fe} * \text{Co} * \text{fathead} * \text{NV} * \text{Dai}) * \exp(\text{-TD/Day}) \quad (1)$$

Where Fe, fever; Co, cough; fathead, fatigue and headache; NV, nausea and vomiting; Dai, diarrhea. The first term in Eq. (1) is total influence of primary symptom and second term increases the primary symptom per day as the number of day's increases.
R0(Day) the secondary symptom caused per day is given by
Eq. (2)

$$R0(Day) = 1 - \exp(-\text{Primary(Day)}) \quad (2)$$

The Model of SIMBO-T is shown in Figure 7. It basically works in two steps i.e. Evaluate Health and Treatment. The fitness function is evaluated which determines the heath of each individual the treatment which is based on the symptoms of patient is done initially with standard amount of medicine/dose. This amount of dose can be increased or decreased based on the response of patient and it also depends on primary and secondary symptom as well as current health and pandemic health. If the amount of dose is given/adjusted over time based on the response of the patient/individual the optimal results can be achieved more festally [1].

.

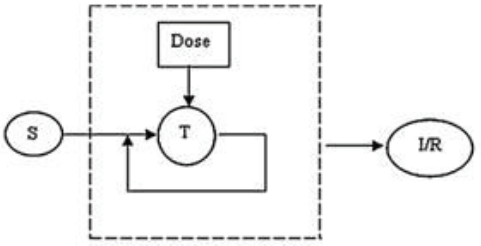


Fig 7. Model of SIMBO-T.

The equation for the dose/medicine given to the individual is shown as

$$\begin{aligned}\text{Dose}(m+1) = {} & \text{Dose}(m) * \text{Md} + \text{Primary (Day)} * \text{rand} * \\ & (1 - \text{Current\_health}(m)/\text{rand} * \text{PH}) \\ & + \text{R0 (Day)} * \text{rand} * (\text{Current\_health}(m) - \text{PH})\end{aligned} \quad (3)$$

The change in the individual state is given as

$$S(m+1) = S(m) * \text{Ms} + \text{Dose}(m+1) \quad (4)$$

IV. FITNESS FUNCTION

A fitness function is the ratio of the number of covered nodes of the path of the target node to the total number of nodes of the path of the target node. Each test case is feed to the program for execution and all the nodes covered by this test case are recorded which is called execPath. The fitness value $ft(\text{Vft}$ for each Individual/chromosome/particle v_i (i = 1, …, S) in a population having size 'S' is calculated as follows:

1. Determine execPath: find set of nodes covered by a test case.
2. Determine path(n): path of the target node
3. Find (path(n) - execPath): Identify nodes/node which are not covered in path
4. Find (path(n) - execPath)$'$: Identify covered nodes/node in path
5. Find | path(n) – execPath$'$ | : count number of nodes covered in the path.
6. Find |path(n)| : count number of nodes of the path of the target node

Then

$$ft(vi) = \frac{|(path(n) - execpath)'|}{|path(n)|} \quad (5)$$

The individual/ particle/chromosome is represented by test case and a test case v_i is optimal if its fitness value is equal to 1 i.e. $ft(v_i)$ = 1 [29]. The fitness value is only method of feedback to optimization algorithms like GA, SIMBO, PSO etc.

V METODOLOGY

The proposed software is developed using C language and MatLab. First test object source code is fed to program for instrumentation. The instrumented program provides the information for the covered node. Then, the instrumented program is fed to proposed search algorithms. The proposed search algorithm technique besides this instrumented program also accepts a file containing information about leaves of spanning tree along with their paths as inputs. It accepts other parameters like population size, length of chromosomes, maximum number of generations, and probabilities of the crossover and mutation operators in case of GA. The parameters in case of PSO are like number of agents, maximum iteration, dimensions etc.

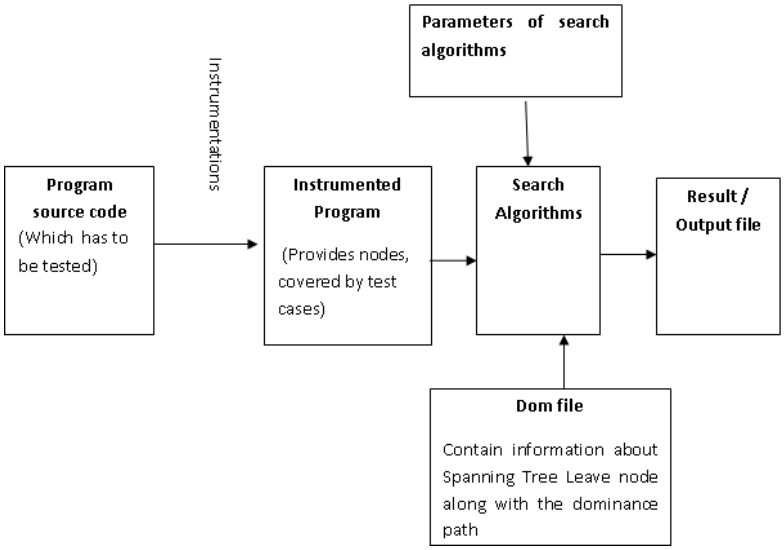


Fig. 8. Building Blocks of Automatic Test Data Generator Program

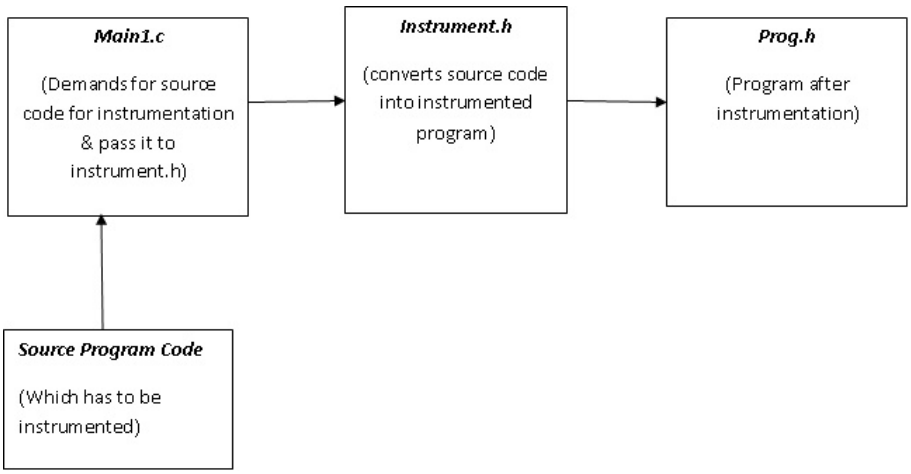


Fig. 9. Steps of Instrumentation

VI. EXPERIMENTAL RESULTS AND CONCLUSION

Experiment is done on commonly used computer programs for testing the performance of newly proposed approach using test data generation techniques like SIMBO, GA and PSO for both dominance concept and Spanning Tree. The performance of all three optimization methods SIMBO, GA and PSO is evaluated using both concepts dominance Tree concepts as well as Spanning Tree concept.

From the table II and figure 10. it is clear that performance of SIMBO is better than other in number of generation taken by algorithm which is directly proposition to the time taken by algorithm to search, so SIMBO is better than other. Further it is clear the number of generations required in using spanning tree concept is less as compare to dominance tree concept. Again here performance of SIMBI is better than PSO which is better than GA in both concepts.

TABLE I : COMPARISON IN TERM OF NUMBER OF GENERATION

	No of Generation					
	Spanning Tree Concept			*Dominance Tree Concept*		
Prog No.	GA	PSO	SIMBO	GA	PSO	SIMBO
1	18	14	9	19	15	11
2	6	3	2	7	4	3
3	11	7	3	13	9	5
4	5	5	2	6	5	4
5	11	9	6	12	11	7
6	18	14	9	21	16	10
7	8	5	4	10	6	5

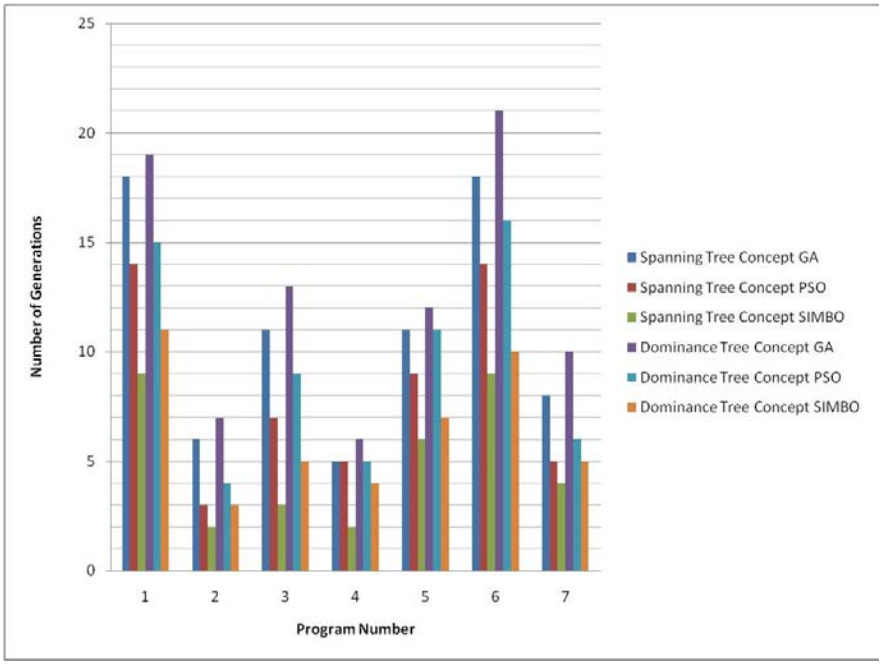


Fig. 10 Evaluation in term of number of Generations taken by SIMBO-T, GA and PSO

From the Figure 11 and Table III , it is clear the Using spanning tree concepts the coverage percentage ratio is near by 100 or best as compare to the concept of dominance tree, further the percentage of converge ratio of SIMBO is better as compare to PSO and GA in both concepts (Dominance Tree and Spanning Tree).

TABLE II : COMPARISON IN TERM OF COVERAGE RATION PERCENTAGE

Prog No.	*Percentage of Coverage Ratio*					
	Spanning Tree Concept			*Dominance Tree Concept*		
	GA	PSO	SIMBO	GA	PSO	SIMBO
1	100	100	100	100	100	100
2	98	100	100	95	100	100
3	100	100	100	100	100	100
4	100	100	100	100	100	100
5	100	100	100	100	100	100
6	95	98	100	85	94	100
7	89	97	100	73	91	99

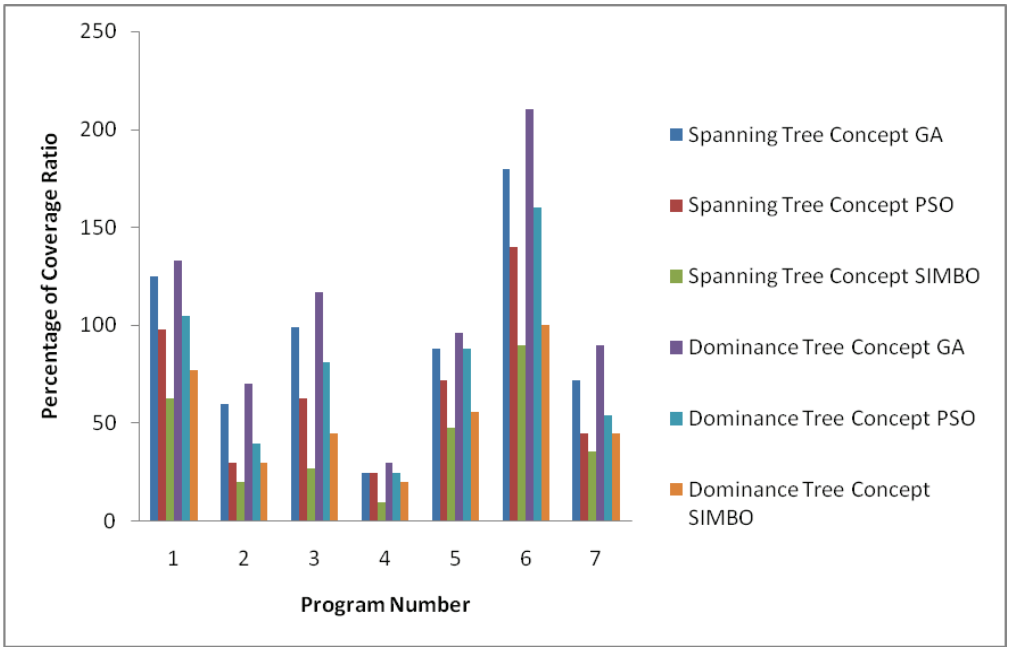


Fig. 11 Evaluation in term of coverage ratio percentage by SIMBO-T, GA and PSO

By analysis of figure 12 and Table IV, the number of test cases generated in Spanning Tree concept is less as compare to Dominance tree. The number of test cases generated in SIMBO in both concept as compare to other two optimization method i.e. GA and PSO. As SIMBO-T achieves 100% coverage in less number of iterations as compare to GA and PSO which concludes that test cases generated by SIMBO-T are unique as compare to PSO and GA. Overall, it can concluded that Spanning tree concepts is better than Dominance tree in all aspects i.e. in terms of time consumption, generation of unique test cases, coverage ratio percentage, number of generations etc. Further, it can be concluded that SIMBO-T performs excellent as compare to other two algorithms i.e. PSO and GA in both concepts.

Table III : Comparison in term of number of test cases generated

	No of Test cases					
	Spanning Tree Concept			*Dominance Tree Concept*		
Prog No.	GA	PSO	SIMBO	GA	PSO	SIMBO
1	125	98	63	133	105	77
2	60	30	20	70	40	30
3	99	63	27	117	81	45
4	25	25	10	30	25	20
5	88	72	48	96	88	56
6	180	140	90	210	160	100
7	72	45	36	90	54	45

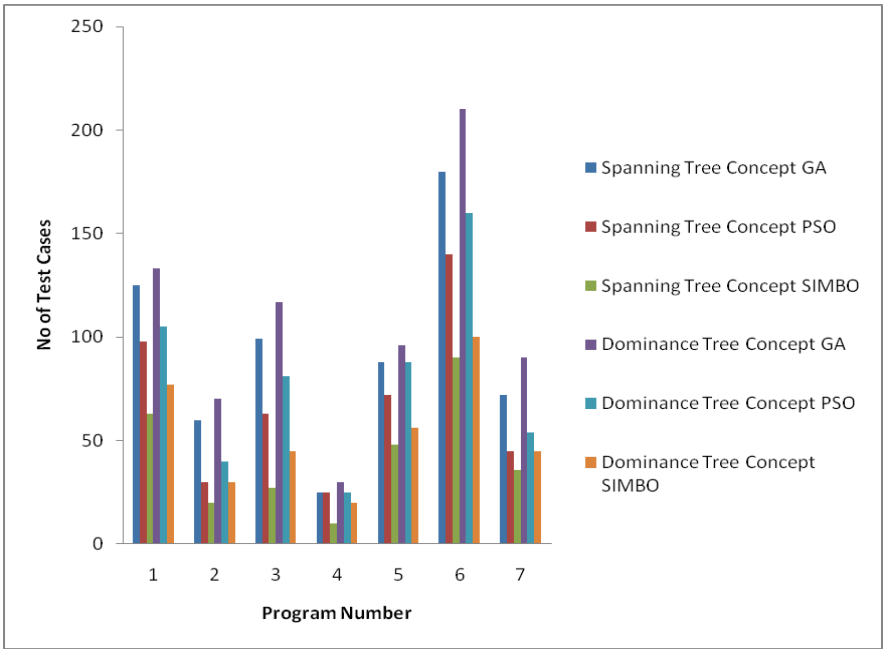


Fig. 12. Evaluation in term of number of test cases generated by SIMBO-T, GA and PSO

References

[1] S. S. Pattnaik, K. M. Bakwad, B. S. Sohi, R. K. Ratho and S. Devi "Swine Influenza Models Based Optimization (SIMBO)", Journal of Applied Soft Computing, Vo. 2013, pp. 628-633.
[2] B. Beizer, "Software Testing Techniques", Second Edition, Van Nostrand Reinhold, NewYork, 1990.
[3] R. A. DeMillo, and A. J. Offlut, "Constraint-based automatic test data generation", IEEE Transactions on Software Engineering, Vol. 17, No. 9, pp. 900-910, 1991.
[4] S. Desikan, G. Ramesh, "Software testing principles & practices", Pearson Education.
[5] R. Boyer, B. Elspas, and K. Levitt, "Select-a formal system for testing and debugging programs by symbolic execution", SIGPLAN Otices, Vol. 10, No. 6, pp. 234-245, June 1975.
[6] L. Clarke, "A system to generate test data and symbolically execute programs", IEEE Transaction on Software Eng., Vol. SE-2, No. 3, pp. 215- 222, Sept. 1976.
[7] C. Ramamoorthy, S. Ho, and W. Chen, "On the automated generation of program test data", IEEE Trans. Software Eng., vol. SE-2, no. 4. pp. 293-300,

Dec. 1976.
[8] W. Howden, "Symbolic testing and the DISSECT symbolic evaluation system", IEEE Trans. Software Eng., Vol. SE-4, No. 4, pp. 266- 278, 1977.
[9] J. H. Holland, "Adaptation in natural and artificial system, Ann Arbor", The University of Michigan Press, 1975.
[10] D. Ince, "The automatic generation of test data", Computer Journal, Vol. 30, No. 1, pp. 63-69, 1987.
[11] W. Miller and D. Spooner, "Automatic generation of floating-point test data", IEEE Trans. Software Eng., Vol. SE-2, No. 3, pp. 223-226, Sept. 1976.
[12] J. Offutt, Z. Jin, and J. Pan, "The Dynamic domain reduction procedure for test data generation", Software Practice and Experience, Vol. 29, No. 2, pp. 167–193, January 1997.
[13] N. Gupta, A. P. Mathur, and M. L. Soffa, "Automated test data generation using an iterative relaxation method", In ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering (FSE-6) Orlando, Florida, pp 231–244, November 1998.
[14] M. R. Girgis, "Automatic test data generation for data flow testing using genetic algorithm", Journal of Universal Computer Science, Vol. 11, No. 6, pp. 898–915, 2005.
[15] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information", IEEE Transactions on Software Engineering, Vol. 11, No. 4, pp. 367-375, 1985.
[16] F. E. Allen and J. Cocke, "A program data flow analysis procedure", Communication of the ACM, Vol. 19, No. 3, pp. 137–147, 1976.
[17] J. Kennedy and R. Eberhart, "Particle swarm optimization", IEEE International Conference on Neural Networks, IEEE Press, pp. 1942–1948, 1995.
[18] A. Windisch, S. Wappler and J. Wegener, "Applying paricle swarm optimization to software testing", ACM, GECCO, London, England, United Kingdom, New York, pp. 1121-1128, 2007.
[19] K. Agrawal and G. Srivastava, "Towards software test data generation using discrete quantum particle swarm optimization", ISEC, Mysore, India, pp. 65-68, February 2010.
[20] A. Li and Y. Zhang, "Automatic generating all-path test data of a program based on pso", World Congress on Software Engineering. IEEE, Los Alamitos, pp. 189-193, 2009.
[21] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory", 6th International Symposium on Micromachine Human Science, pp. 39–43, 1995.
[22] N. Narmada and D. P. Mohapatra, "Automatic Test Data Generation for data flow testing using Particle Swarm Optimization", Communications in Computer and Information Science , Vol. 95, No. 1, pp. 1-12, 2010.
[23] A. S. Andreou, K. A. Economides and A. A. Sofokleous, "An automatic software test-data generation scheme based on data flow criteria and genetic algorithms", 7th IEEE International Conference on Computer and Information Technology, pp. 867-872, 2007.
[24] R. P. Pargas, M. J. Harrold and R. R. Peck, "Test Data Generation using Genetic Algorithms", Software Testing Verification and Reliability, Vol 9, pp. 263-282, 1999.
[25] C. C. Michael, G. E. McGraw and M. A. Schatz, "Generating software test data by evolution", IEEE Transactions on Software Engineering, vol. 27, no.12, pp. 1085-1110, 2001.
[26] A. S. Ghiduk, M. J. Harrold and M. R. Girgis, "Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage", 14th Asia-Pacific Software Engineering Conference, 2007.
[27] J. T. Alander, T. Mantere, and P Turunen, "Genetic Algorithm Based Software Testing", In Proceedings of International Conference (ICANNGA97), Wien, Austria, pp. 325-328, April 1998.
[28] A. Khamis, R. Bahgat and R Abdelaziz, "Automatic test data generation using data flow information", Dogus University Journal, 2, pp. 140-153, 2000.
[29] A. S. Ghiduk and M. R. Girgis, "Using Genetic Algorithms and dominance concepts for generating reduced test data", informatics, 34, pp. 377-385, 2010.
[30] K. H. Chang, J. H. Cross, W. H. Carlisle and D. B. Brown " A framework for intelligent test data generation", journal of intelligent and robotic systems-theory and application, Vo. 5, No. 2, pp. 147-165, 1992.
[31] D. G. Jadhav, S. S. Pattnaik and S. Das "Memetic Algorithm with Local Search as Modified Swine Influenza Model-Based Optimization and Its Use in ECG Filtering", Journal of Optimization, Vo. 2014.
[32] B. J. Coburn, B.G. Wagner and S. Blower "Modeling influenza epidemics and pandemics: insights into the future of swine flue (H1N1)", BMC Medicine (2009), pp. 7–30.
[33] A. Biswas, K. K. Mishra, S. Tiwari, and A. K. Misra, "Physics-inspired optimization algorithms: a survey," Journal of Optimization, vol. 2013, Article ID 438152, 16 pages, 2013. View at Publisher · View at Google Scholar.